Federated Architectures Lecture 11

Fault Tolerant Systems (II)

Today

Fault-Tolerant systems in practice

Rubber meets the road, no fluff here!

- Non-Stop (Tandem)
- Targon32 (Seimens-Nixdorf)
- Hive (Stanford)

Themes:

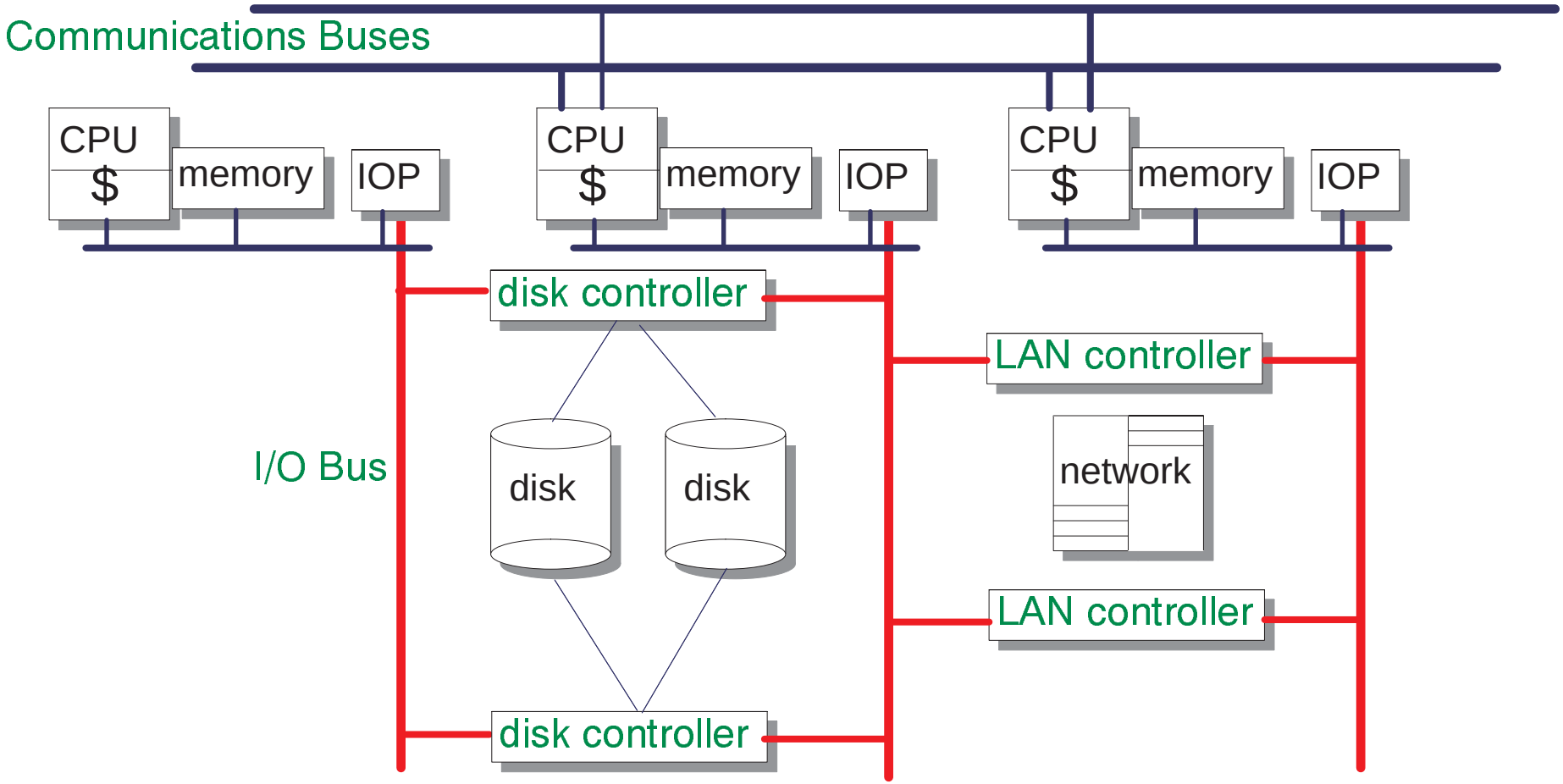
- Tolerance vs. Containment
- Performance vs. Tolerance
- Sharing vs. Isolation
- Way too much stuff for today!

Tandem Non-Stop

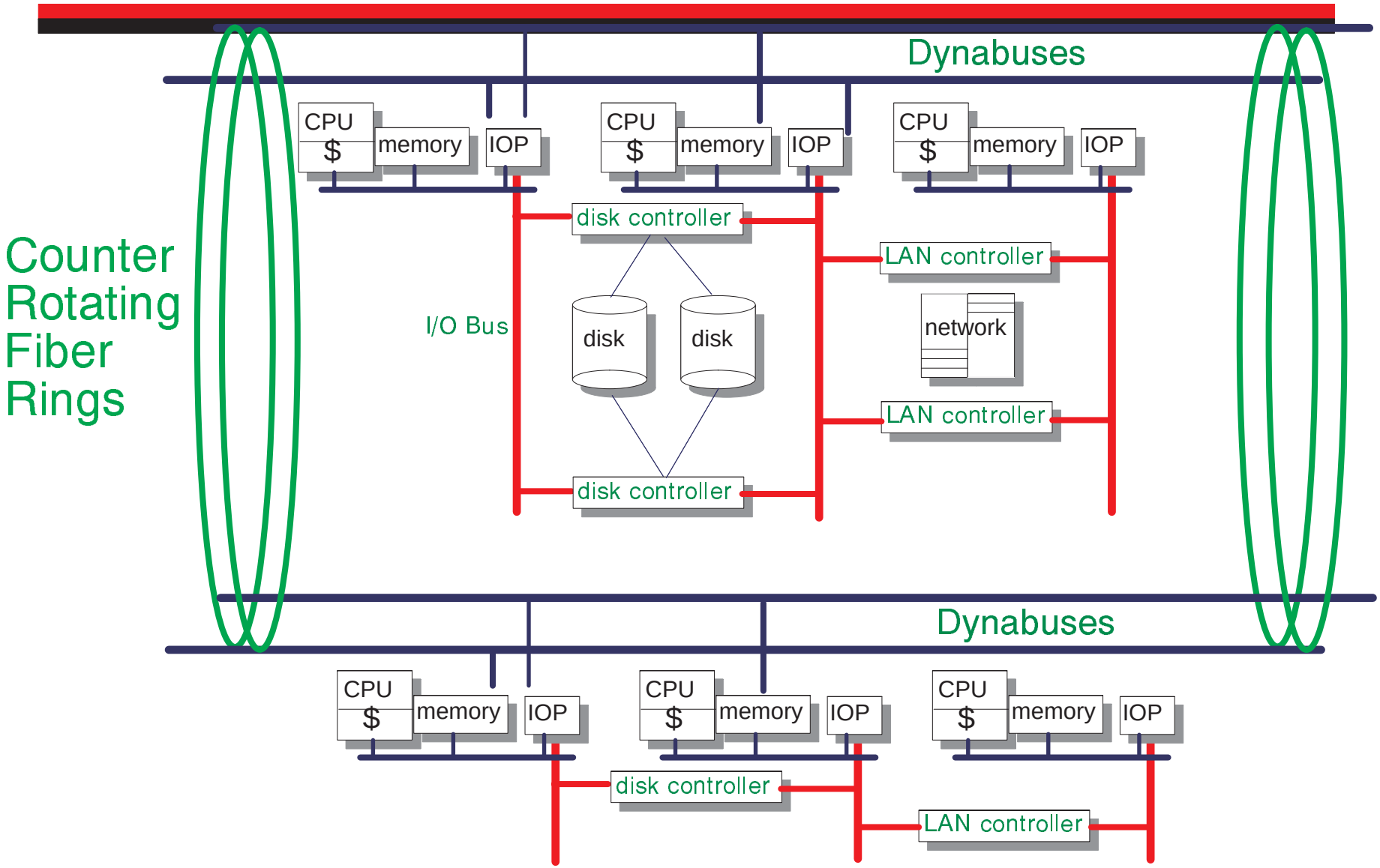
Big ideas:

- Replication = instant MTTR
- Transactions key to Fault-Tolerance
  - start, end, abort, redo
  - If there is a failure sometime before end, can redo the transaction
- Transactions provided for by the OS
- Replication also provides Scalability

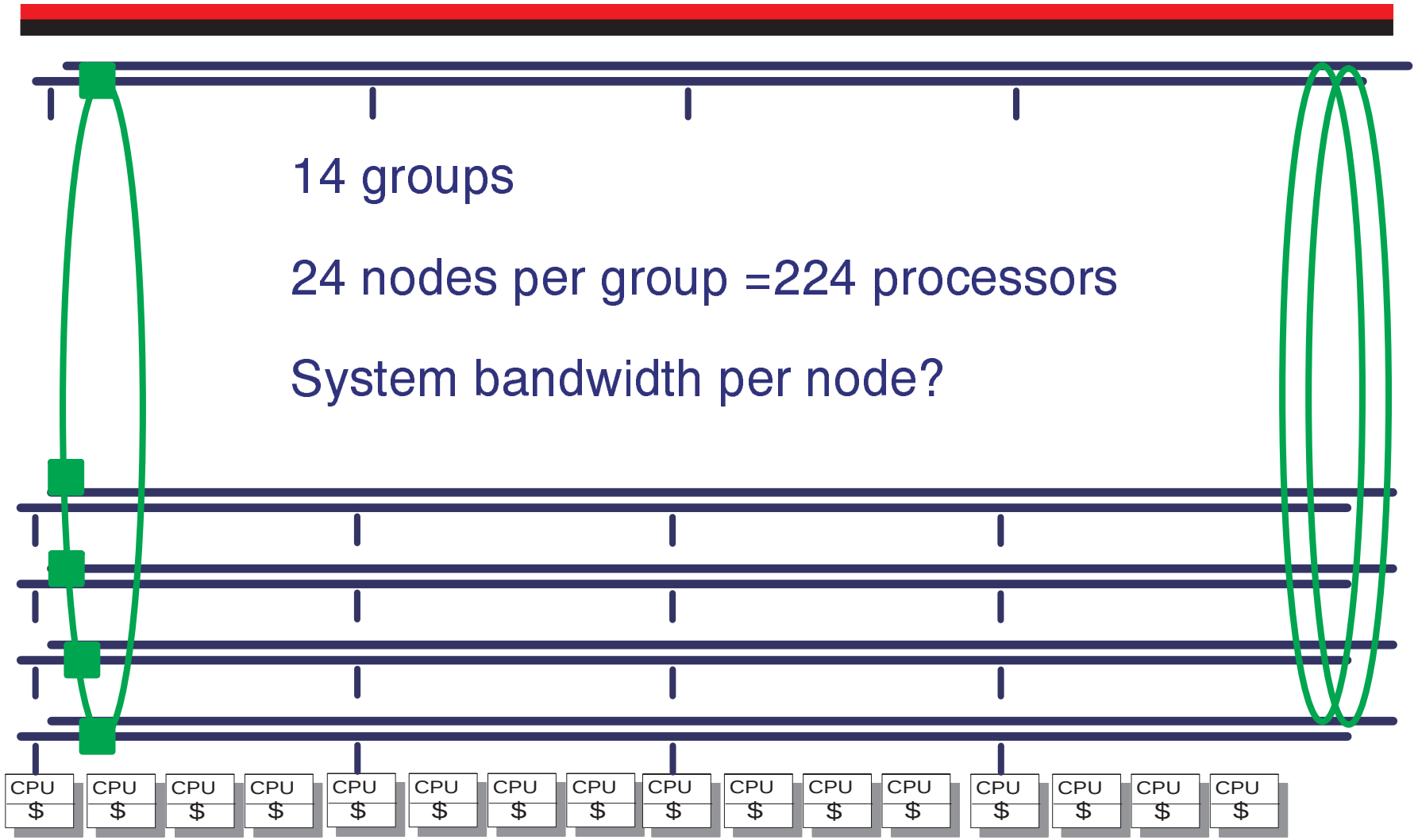
Basic NS Architecture



Extended NS Architecture



Tandem Scaling



NS Network (FOX)

Why Rings?

- More FT than a single switch
- Reduces cables
- Easier to build than switching arch
- "fair" properties under load (compared with CSMA/CD protocols)

Single ring has little FT

- Solution: two pairs of rings
- A pair is counter rotating like FDDI?
  - If one breaks, loop back, create a single ring?

Guardian: A μ-kernel?

Kernel executive

- address spaces
- scheduling
- messages
- time
- synchronizartion primitivies
- debugging

User-level servers for

- Virtual Memory
- Filesystem
- I/O process
- Disk process
- Network
- Transaction Manager

OLTP OS (Guardian)

Transactions wind their way through the whole system

- Unwind transaction
- "Garbage collection"?

Transaction advantage: random updates turned into sequential updates

- write transaction to the log
- return transaction complete
- update disk structure of DB when it's convenient

Alive Protocol

- Every 2 seconds, ping all other processors.
  - Also check your set, did you hear from everyone in the last interval?
  - A missing processor indicates a missing member from the group
  - Enter regroup protocol

Alive Protocols

- Tandem IMA protocol
- Porcupine's TRM
- Targon Crash detection scheme
- Very similar:
  - Set of "pingers", in a ring
  - One node suspects a failed node, becomes the leader
  - Broadcast a re-group to all
  - Nodes peform a "barrier"
    - Two phased to make sure
    - Why two phased?
  - New membership is broadcast from the leader

Alive protocols:1000 nodes?

- Would an alive protocol scale to 1000 nodes?
  - Assume a MTTF of 45 days

$$MTTF\ 45days = Pf_{\text{sec}}(0.00000154)$$

$$1000 \times Pf_{\text{sec}}(0.00000154) \approx Pf_{cluster}(0.00154)$$

$$\frac{1}{Pf_{cluster}} = MTTF_{cluster}$$

1 node failure every 65 seconds

Modeling vs. Reality

Previous model seems too pessimistic

- MTTF could be longer
  - Single node MTTF of 300 days -> 432 sec cluster MTTF
    - Some machine fails every 7 hours
    - Much more believable
- Model could be wrong

Assuming because memoryless process we can scale from days to seconds is probably an invalid assumption

Alive protocols

- How painful is the regroup protocol?
  - Three rounds of broadcast
    - How painful is broadcast? (on 1000 nodes)
    - Natural tree structure
    - How to make a broadcast tree FT?
- Heirarchical membership protocols needed?
- Loose membership algorithms?

Short project

- What is the MTTF of real machines?
  - Very easy to compute!
    - E.g. "last reboot" on Solaris, Linux
- Is it fractal (not possion)?
    - No reboot in period means unlikey to reboot
    - reboot means highly likely to reboot in future
- Does usage make a difference?
  - e.g. ugrad machines (cereal) vs. fac. vs grad.
  - Production vs desktop
- How to get this information on NT/98?

Time Synchronization

Another similarty to porcupine

Important result: Possible to synch to within half an RTT (one way time)

In practice (e.g xntpd) lore says its possible to get to within 2-3 msec of each other

- Is this good enough?

Scalability through time

How can the system absorb load.

- As we increase load, what happens?
- Typical graph: some saturation point
- Assumes can't increase size of the system to support load
- Through what time-scales can a system scale and still remain available?
  - Minutes
  - hours
  - days
  - years
  - decades

Traditional HA systems

- Apply a process-pair idea to the whole system
- Using a DB, the basic idea is to send the log to the remote site
- Log replay used to recover the system state
- Multiple variations on this theme
  - One system on standby only
  - both systems recover for each other
  - a single coordinator site

FT Unix (Targon)

- Tolerates hardware and system faults via repliaction
- Similar hardware arch to Tandem NS
- μ-kernel like architecture too
  - But idea was transparent Unix, not OLTP based OS
- Bus based networking
- Regroup protocols

Targon μ-kernel design

Familiar design

- File server processes
- VM (paging) servers
- Network (tty)
- Disk processes
- Process ID servers
  - Special process maps u-kernel to unix pid space?
  - Compare to QNX

How to support a global PID namespace?

- Special process
- Broadcast PID mappings
- "broadcast" PID mapping algorithm

Bus Protocol

- Assert bus request
  - Arbitration protocol?
- broadcast 3 destinations
  - destinations can NACK
  - Retry protocol if NACKed?
- No nacks in timeout window: broacast on bus
  - All processors listen, can pick up message if a destination

Process Pair Architecture

- Same idea as tandem process pair
  - but messages, not transactions, are the recovery unit
- μ-kernel architecture forces all interactions to operate via messages
  - What about signals?
- Primary runs normally
  - Messages sent to process B go to B's backup (B') and A'

Message history

- Assume process is deterministic
  - What if messages sending was based on a random seed?
- On takeover, backup runs "count until sync" messages
- At the count, know that the processes should be in the same state
  - What about ...
    - Signals
    - Non-deterministic servers (file server process)

Process backup classes

- None
- Quarterbacks
  - Backup until crash
  - On machine restore, do not rebackup
- Halfbacks
  - backup until crash
  - restore backup when node reboots
- Fullbacks (not implemented)
  - on crash, create a new backup
  - can use same mechanism to support migration and load balancing

Performance

Interesting comparison (but very brief!)

- μ-kernel design reduces overall performance by 10-15%
  - Some kernel functions reduced by 20%
- Is this consitent with other results (Mach, QNX, Hurd...)
- Is this an acceptable performance degradation?

What about HW costs?

- Need a special purpose bus
- Huge engineering effort
- Possible to do with more standard components?

All that work...

What does Targon really cover anyway?

- Single-point hardware faults
- Heisnbugs in the OS (common?)
  - Long term bugs in the OS

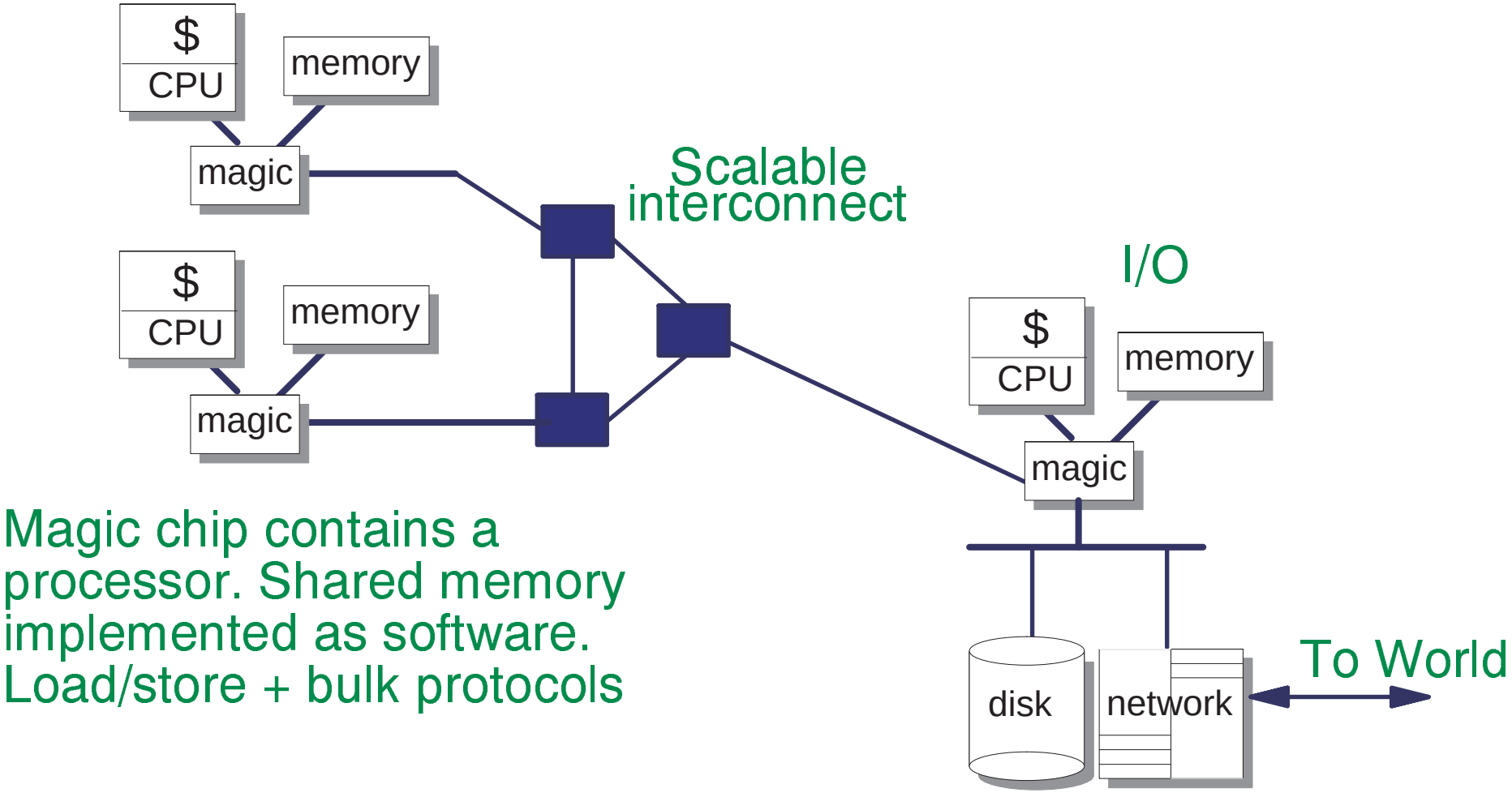
What about application errors?

- Does not cover heisnbugs in application
- Deterministic execution: no application "rollback" possible
- Need the notion of a "transaction" inside the application
  - Is the guardian OS the right approach?

Hive

- Big idea: Fault containment vs fault tolerance
- Fault tolerance requires extra resources
  - What if you're app uses all machine resources? Can't be fault tolerant!
- Add a new level of abstraction: the cell
  - A Cell is the unit of fault containment
  - Cell crashes should not toast other cells
- If sucessful, SMP no worse than a cluster?
  - But get the pro's of hardware shared memory

Flash Hardware



Magic chip contains a processor. Shared memory implemented as software. Load/store + bulk protocols

Fundamental tradeoff

Hive researcher quote:

" I'm have to work to convince these shared memory hardware guys to add messaging passing hardware".

- Why do we need message passing?
  - Message model allows fault - containment
  - Catch faults at message boundary
  - How to build a fault containment shared memory model?
    - load/store boundary?
    - Hive punts uses messaging for containment

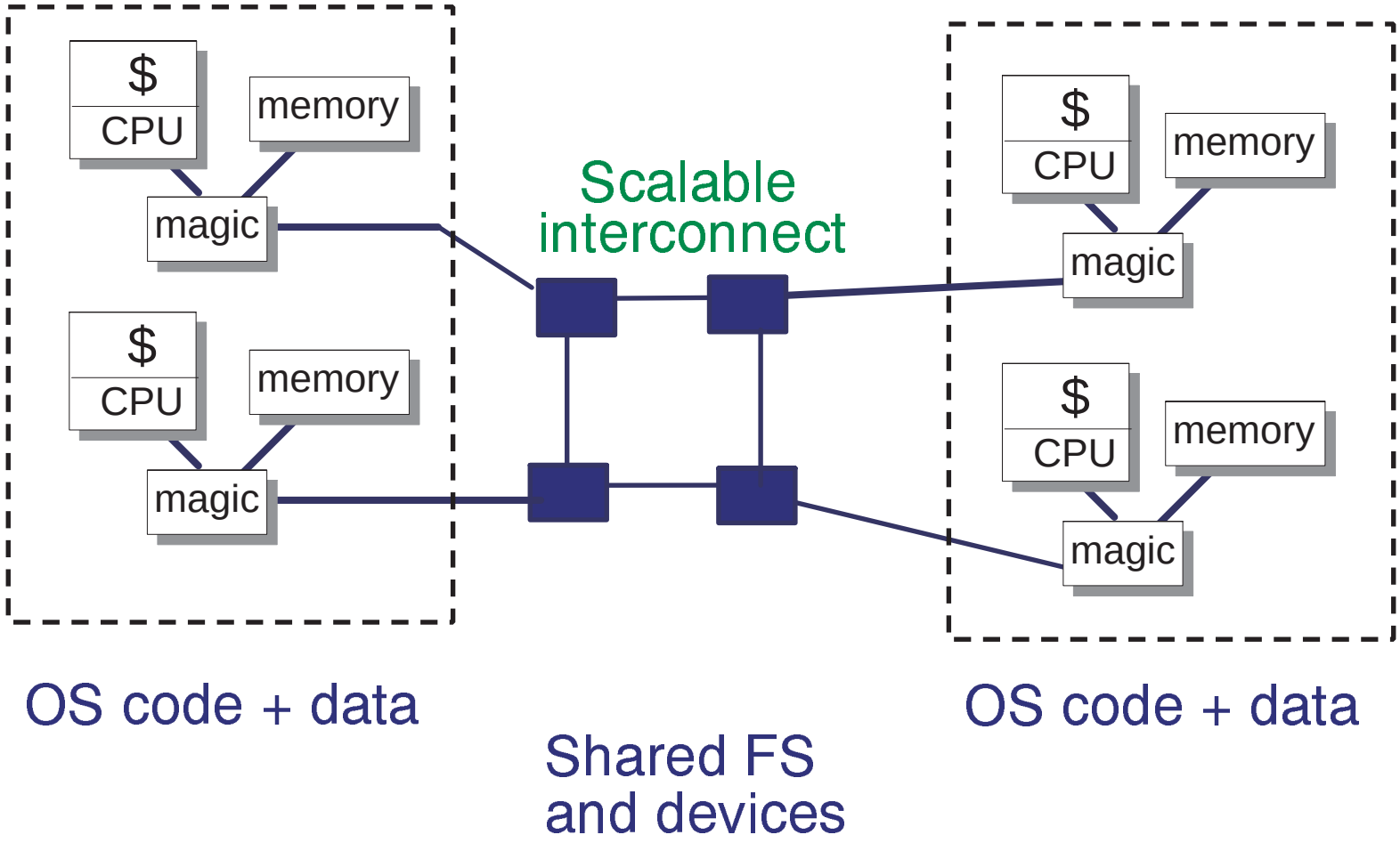
Same problem with the E10K!

- Fault containment = cluster arch.

Hive Recovery

- Basic recovery scheme is same as Tandem, Targon, Porcupine
  - Global regroup upon detection of node failure
  - Hive actually punted on implementation
    - "Oracle" gives current group status
  - Requires "broadcast"

Cell Abstraction



Sharing memory

Two kinds of page-level sharing

- logical level sharing
  - A remote process maps in a page from a remote cell
  - Operates at the process level
- Physical level sharing
  - Remote cell OS's borrows the physical page owned by a cell

Why have two kinds?

- how to share non-process pages?
  - buffer cache, OS structures?
  - Would a μ-kernel design eliminate the need for physical shareing?

Fast RPCs in flash

Presented as an afterthought

- but a fundamental technology
- 50 μsec RTT for a page fault RPC
  - includes interrupting the processor
  - Close to fastest messaging abstractions
  - But more functionality ...

- Interrupt service and delayed RPCs
  - Service at IRQ level or
  - queue and service at a lower-level

Flash Performance

SMP programs: no effect

- But little use of OS services sans mapping in pages
- better not have an effect or something is very broken

Pmake

- Uses OS
- 11 % slowdown
- Same class of slowdown as Targon system
- Is fault containment worth 10%?
- Would an μ-kernel be better?

Trust Nothing Architecture

- TNA ideas:
  - Reduce chance for a major disaster in system software and hardware
  - Build system out of multiple OSes and Hardware:
    - Linux/Solaris OS combo
    - Sparc x86 node hardware combo
    - Four types of nodes!
      - Endian problems
    - A silly idea in general? More trouble than it's worth?
    - How often does a meltdown occur?

Next Week

- No Class next week
  - Send progress reports on projects!
  - What's implemented
  - How much longer you think it will take